AD-A087 521 KRANNERT GRADUATE SCHOOL OF MANAGEMENT LAFAYETTE IN F/G 12/1
A MODAL LOGIC FRAMEWORK FOR AN A.I. PLANNING SYSTEM.(U)
JUL 80 R P MCAFEE, A B WHINSTON DAAG29-79-C-0154
UNCLASSIFIED ARO-16231.4-EL NL

1 OF 1
AD
A087 521

END
DATE
FILMED
9-80
DTIC

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) (18) ARO 16231.4-EL

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A087521	3. RECIPIENT'S CATALOG NUMBER (19)
4. TITLE (and Subtitle) (6) A MODAL LOGIC FRAMEWORK FOR AN A.I. PLANNING SYSTEM.		5. TYPE OF REPORT & PERIOD COVERED (9) Technical rept.
		6. PERFORMING ORG. REPORT NUMBER Contract No. DA79C0154
7. AUTHOR(s) (10) R. Preston McAfee and Andrew B. Whinston	(12)	8. CONTRACT OR GRANT NUMBER(s) (15) DAAG29-79-C-0154
9. PERFORMING ORGANIZATION NAME AND ADDRESS Purdue University Krannert Graduate School of Management West Lafayette, IN 47907		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Research Office Post Office Box 12211 Research Triangle Park, NC 27709		12. REPORT DATE (11) Jul 1980
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) (12) 34 LEVEL II		15. SECURITY CLASS. (of this report) unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

DTIC ELECTE C

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official department of the Army position, policy, or decision, unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A formal development of planning systems is constructed in this paper. The problem solving system is developed in the framework of a modal logic and the concept of weakest precondition is introduced to guide the solution procedure. The procedure is a generalization of several well-known priblem solving algorithms. Moreover, there may be some gains in practicality of the algorithm resulting from this extension, which is illustrated by an example.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601 | 404200 Jm

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ADA087521

80 8 1 057

A MODAL LOGIC FRAMEWORK FOR AN A.I. PLANNING SYSTEM *

R. Preston McAfee

and

Andrew B. Whinston

Krannert Graduate School of Management

and

Department of Computer Science

Purdue University

* The authors gratefully acknowledge the theoretical and editorial assistance of Fran Berman and Mike Odonnell in the preparation of this paper. Any errors contained herein are the responsibility of the authors.

Abstract

A formal development of planning systems is constructed in this paper. The problem solving system is developed in the framework of a modal logic and the concept of weakest precondition is introduced to guide the solution procedure. The procedure is a generalization of several well known problem solving algorithms. Moreover, there may be some gains in practicality of the algorithm resulting from this extension, which is illustrated by an example.

I. Introduction

In this paper, we consider formalizing problem solvers for predicate calculus expressions. Many of the problem solving systems in the literature, while often ingenious, are essentially variations on the Resolution Principle [5]. In an ad hoc way, most of these have introduced operations over predicate calculus models, and devised some method of machine search over the operators. It is precisely these features of the literature that are dealt with in this paper.

Our first action is to demonstrate that STRIPS [2] programs are examples of programs in the formal language Dynamic Logic [3]. This is a useful feature in itself as it allows us to examine STRIPS with the analytical tools a formal language provides. We then can observe that our operators are not limited to STRIPS type operators. A much larger class of operators becomes available via this formalization. These operators may simulate actions, as in STRIPS, or may only be calculations, such as multiplication.

An important advantage of a formal theory of operators is the analytical tools available. In this paper we focus on Dijkstra's [1] weakest preconditions as an excellent example of operator analysis. These weakest preconditions permit us to evaluate in advance whether application of an operator will achieve a goal. Because this is a formal criteria, it is naturally embeddable into a machine for automated decisions. Another useful feature of the formal analysis is the ability to prove results about the system. In particular, it was possible that the STRIPS approach, involving only concatenations of operators and not allowing for unions or iterations, was a restriction of a system involving these latter operations. In this paper, we prove that concatenations of operators is sufficent for solving STRIPS type problems. The evaluation function [2] of STRIPS is an ad hoc method of determining whether or not a given operator is appropriate. We formalize this by defining a measure of distance, called a pseudometric, over the space of predicate calculus models. In most cases this allows for increased efficiency in the decision process. Since this simulates the approach of a human decision maker, it is a useful tool in mechanized problem solvers. Humans have some notion of what models are close to one another, and formalizing this may allow the machine a similar faculty. Moreover, pseudometrics have been implicitly assumed in many of the extant problem solvers. There is some advantage to their explicit use, as shown in McAfee and Whinston [4].

II. .1 The Guarded Commands Language

In order to extend and formalize STRIPS, we may embed STRIPS inside a larger formal language. Although STRIPS operates within the context of predicate calculus, STRIPS is essentially beyond predicate calculus, because its operators are not tautological proof procedures. STRIPS operators are modal operators, i.e. their application reflects a change of states. For this reason it is sensible to embed STRIPS in a language capable of expressing facts abouts state changes. As we shall see, a subset of Dynamic Logic [3] called the Guarded Commands language [1] makes an excellent choice for such a language.

The guarded commands language is a strict extension of predicate calculus, and the set of terms is identical in both. The set of programs, PGC, for Programs in the Guarded Commands language, are defined inductively as follows. We have restricted the origional formulation to deterministic programs. Let e be any term, P and R be predicate calculus formulas, and x a variable. Then

1) ABORT $\in$ PGC

2) $x \leftarrow e \in$ PGC, (assignment)

3) for α, β, $\in$ PGC, $\alpha;\ \beta \in$ PGC (concatenation)

4) for α, $\beta \in$ PGC and predicate calculus expression P,

$P?;\ \alpha \cup \neg P?;\beta \in$ PGC (program union)

5) $(P?;\ \alpha)^{*};\neg P? \in$ PGC (unbounded iteration)

This is actually a subset of Dijkstra's original program set, limited to deterministic programs. Determinism is a useful assumption that will cost little , if any, generality.

The programs are intended to be interpreted with variables ranging over the real numbers and the standard meaning of any functions and constants necessary, e.g. +, . , 0, 1. We assume there is a well defined interpretation of predicate symbols. The predicate calculus quantifiers and connectives are all given their standard interpretations. The semantics of the program induction are given as follows:

1) Abort

2) Assign term e to variable x

3) do α then β

4) If P do α else β

5) While P do x

Observe that do loops may be constructed using iteration:

$x \leftarrow o;\ (x \leq n?;\alpha;\ x \leftarrow x\ +1)\ ^{*}\ ;\ x > n?$

is a program which executes α for $x = 0, . . . n$

For a more specific example, consider the PGC program

$i \leftarrow 1;\ j \leftarrow 0;\ (i \leq n?;\ j \leftarrow j + m;\ i \leftarrow i + 1)^{*}\ ;\ i > n)?$

This program adds m to j n times, that is, it calculates mn. Actually, in the STRIPS environment, we need only begin with a *finite set* $\underline{\bar{X}}$ *of* programs (operators) from which we may compose new programs. This forms the set of Elementary Programs E , which is defined inductively below:

1) $\underline{\bar{X}} \subsetneq E$

2) $\alpha, \beta, \in E$ then $\alpha;\ \beta \in E$

3) $\alpha, \beta, \in E$ and P a first order statement, then $(P?;\ \alpha \cup \neg P?;\ \beta) \in E$

4) $\alpha \in E$ and P a first order test then $(P?;\alpha)\ ^{*};\ \neg P \in E$

The semantics are the same as in PGC.

Given a set of programs E , a sub-language of DL, called PDDL-GC, is defined similar to the predicate calculus induction:

1) any predicate calculus expression is in DL

2) for $\alpha \in E$, and DL expressions P, R, then $\neg P$, $P \vee R$ $(\exists x)P$ and $\langle\alpha\rangle P \in DL$

The semantics of these expressions is the same as in predicate calculus except for $< \alpha >$ which is undefined in predicate calculus. $< \alpha >$ P is interpreted to mean "α terminates with P true", that is, after operating α, P will be true.

If α corresponds to a STRIPS operator, then $P \rightarrow < \alpha > R$ means that if P is true, then operation of α will make R true. It is very useful to have such expressions formally expressible, as they may be very useful in guiding a STRIPS search. Through such expressions, we may explicitly represent information on how to procede to make R true. This allows us to replace STRIPS' "evaluation function" [2] with an organized, formally stated and automatable body of knowledge about what works to prove goals. However Dijkstra's formalism possesses considerably more power.

II. .2 The Weakest Precondition

The Guarded Commands Language was designed to make program verification possible and simple. Dijkstra's original purpose was to establish conditions under which a program α will terminate with a predicate calculus expression R true, i.e., what implies $< \alpha >$ R? He demonstrated that in the domain of GC, and hence all subprogram sets, there is a weakest precondition (wp) expressed in infinitary predicate calculus so that $wp(\alpha, R) \equiv < \alpha > R$. Thus we may embed a fragment of the program verification problem inside predicate calculus, where we have the Resolution Principle [5] and other first order theorem provers at our disposal. Dijkstra provides these wp expressions via the induction below, although our notation remains that of Harel [3].

$wp\ (abort, R) \equiv false$

$wp\ (x \leftarrow e, R) \equiv R^e_x$ Where R^e_x is R with x's replaced by e's and all bound variables of R appearing in e are renamed.

$wp\ (\alpha; \beta, R) \equiv wp\ (\alpha, wp\ (\beta, R))$

$wp\ (P?; \alpha \cup \neg P?; \beta, R) \quad (P \rightarrow wp\ (\alpha, R)) \wedge (\neg P \rightarrow wp\ (\beta, R))$

$$wp\ ((P?;\ \alpha)^*;\ \neg P?,\ R) \equiv \bigvee_{n=o}^{\infty} H_n \text{ where } H_o \equiv \neg P \wedge R$$

$$\text{and } H_{n+1} \equiv P \wedge wp(\alpha,\ H_n)$$

Observing that H_n corresponds to the condition implying iteration of the expression (P?; α) n times makes R true, one may quickly establish that all of these equivalences hold. These equivalences provide a method of constructing the weakest precondition implying for $\alpha \in$ PGC, $< \alpha >$ R, for first order R.

In general we cannot expect the weakest precondition of programs involving iteration to have predicate calculus equivalents. $\bigvee_{n=0}^{\infty} H_n$ is an expression in an infinitary version of predicate calculus, and outside of ordinary predicate calculus. The formulation of the weakest preconditions demonstrates that for any program α without iteration, $< \alpha >$ P has a predicate calculus equivalent. We will argue that for a problem solving environment where the initial state is given and fixed, iteration is unnecessary. In this way we will avoid the need to work in an infinitary environment. Clearly, for many types of program analysis, it is necessary to include iteration.

II. 2.3 STRIPS Operators

The operators of STRIPS change world states in an organized fashion. Any STRIPS operator α is given by an ordered triple of sets of predicate calculus expressions $< \{ P_1 . . , P_k\}, \{A_1,. . ., A_m\}, \{D_1,. . ., D_n\} >$. The application of α has a precondition, which is that $P \equiv P_1 \wedge P_2 \wedge . . . \wedge P_k$ holds. If P is true, then operation of α results in a new world state where the expression $A \equiv A_1 \wedge . . . \wedge A_m$ is asserted and $D_1, . . . , D_n$ are no longer asserted. STRIPS presumes that nothing else is changed in the world state transition, except of course that new implications may be derived.

To model STRIPS, we must embed the changes in the truth values of expressions into changes of variables. This observation reveals a direct method of such an embedding, that is, provide for variables whose values are truth values of expressions.

Thus, for each predicate calculus expression E, we define a variable μ_E whose semantic interpretation is

$\mu_E = 1$ indicates E is asserted true

$\mu_E = -1$ indicates E is asserted false

$\mu_E \notin \{1, -1\}$ indicates neither E nor $\neg E$ is asserted.

We may inductively build on this definition

i) $\mu_{\neg E} = {}^{-}\mu_E$

ii) $$\mu_{E \vee F} = \begin{cases} 1 & \mu_E = 1 \text{ or } \mu_F = 1 \\ -1 & \mu_E = -1 \text{ and } \mu_F = -1 \\ 0 & \text{otherwise} \end{cases}$$

iii) $$\mu_{\exists x E} = \begin{cases} 1 \text{ if } \exists x\ \mu_E(x) = 1 \\ -1 \text{ if } \neg(\exists x)\ \mu_E(x) = 1 \\ \text{o otherwise} \end{cases}$$

$\mu_E = x$ is notation for EQUAL (μ_E, x), the equality predicate. We also establish a rule of precedence: If $\mu_E \notin \{1, -1\}$ and it is proved that $\mu_E = 1$, then $\mu_E \leftarrow 1$. This rule of precedence should be used to adjust for new information after any theorem proving, that is, after proving an expression E, make sure to assign $\mu_E \leftarrow 1$. The STRIPS program α is very easy to incorporate into DL with this formalism. Let

$\alpha \equiv P?\ \mu_A \leftarrow 1;\ \mu_{D_1}, \leftarrow 0;\ .\ .\ .\ \mu_{D_n} \leftarrow 0 \cup \neg P?$ ABORT.

Then α tests whether the precondition P holds, if so, it makes $A \equiv A_1 \wedge . . ., \wedge A_n$ true and makes $D_1, . . ., D_n$ unasserted. Otherwise it fails.

A STRIPS system is merely a collection of operators of STRIPS type. In the notation of section 2.1, we let $\underline{\overline{X}}$ = {STRIPS operators written in program form}. The set of admissable programs is then E.

Thus we may embed STRIPS operators into the GC formalism. A STRIPS problem consists of an initial state, expressed in predicate calculus, and a goal. The goal is a predicate calculus expression R. The intent of

STRIPS is to find a sequence of operators $\alpha_1, \ldots, \alpha_m$ so that operation of this sequence results in R being true. This is converted into GC by first observing that a predicate calculus initial model is an assignment of terms to the μ variables. The goal is the same expression R. The problem remains to find any sequence of programs (operators) where wp $(\beta_1; \ldots; \beta_i, R)$ is truein the initial state.

In section 3, we shall develop the mathematics of GC to suit our program synthesis, and not program verification, needs. This will result in the construction of a GC problem solver. In Section 4, we shall return to STRIPS for a comparison.

III . 1 Some Mathematical Results.

In order to show that the planning system in the following section operates in the manner intended, we need some analysis of PDDL-GC. Although many of the sections results apply to other varieties of DL, we will assume PDDL-GC throughout.

Lemma 1: Let α, β be GC programs and $P \equiv wp(\alpha, R)$. Then if there exists a first order test Q? so that $wp((Q?; \alpha \cup \neg Q; \beta), R)$ is true, we have $wp((P; \alpha \cup \neg P?; \beta), R)$ is true.

Proof: $wp((Q?; \alpha \cup \neg Q?; \beta), R) \equiv Q \rightarrow wp(\alpha, R) \wedge \neg Q \rightarrow wp(\beta, R)$ was previously noted. This implies $Q \rightarrow P$, and by contrapositive $\neg P \rightarrow \neg Q$. Therefore $P \equiv wp(\alpha, R) \rightarrow wp(\alpha, R)$ and $\neg P \rightarrow \neg Q \rightarrow wp(\beta, R)$. Consequently $wp((P?; \alpha \cup \neg P?; \beta) R) \equiv P \rightarrow wp(\alpha, R) \wedge \neg P \rightarrow wp(\beta, R)$ is true, as desired.

Lemma 2: (Dijkstra [1]) $wp(\alpha, P \vee Q) \equiv wp(\alpha, P) \vee wp(\alpha, Q)$
$wp(\alpha, P \wedge Q) \equiv wp(\alpha, P) \wedge wp(\alpha, Q)$

Lemma 3: Let $P \equiv wp(\alpha, R)$. Then $wp(P?; \alpha \cup \neg P?; \beta, R) \equiv wp(\alpha, R) \vee wp(\beta, R)$

Proof:
$$\begin{aligned} wp(P?; \alpha \cup \neg P?; \beta, R) &\equiv P \rightarrow wp(\alpha, R) \wedge \neg P \rightarrow wp(\beta, R) \\ &\equiv wp(\alpha, R) \rightarrow wp(\alpha, R) \wedge \neg wp(\alpha, R) \rightarrow wp(\beta, R) \\ &\equiv \neg wp(\alpha, R) \rightarrow wp(\beta, R) \\ &\equiv wp(\alpha, R) \vee wp(\beta, R) \end{aligned}$$

Lemma 3 demonstrates that, under the hypothesis $P \equiv wp(\alpha, R)$, $\cup$ functions exactly like boolean disjunction.

Lemma 4: Let $S \equiv wp(\alpha, P)$ and suppose α terminates.[1]

Then $wp(\alpha;(P?;\beta \cup \neg P?;\gamma), R) \equiv wp(S?;\alpha;\beta \cup \neg S?\alpha;\gamma, R)$

Proof: $wp(\alpha;(P?;\beta \cup \neg P?;\gamma), R) \equiv wp(\alpha, wp(P?;\beta \cup \neg P?;\gamma, R)) \equiv$

$wp(\alpha,\ P \rightarrow wp(\beta,R) \wedge \neg P \rightarrow wp(\gamma, R)) \equiv$

$wp(\alpha,\ P \rightarrow wp(\beta,R)) \wedge wp(\alpha, \neg P \rightarrow wp(\gamma, R)) \equiv$

$wp(\alpha, \neg P \vee wp(\beta, R)) \wedge wp(\alpha, P \vee wp(\gamma, R)) \equiv$

$wp(\alpha, \neg P) \vee wp(\alpha;\beta, R) \wedge wp(\alpha;P) \vee wp(\alpha;\gamma, R) \equiv$

$\neg S \vee wp(\alpha;\beta, R) \wedge S \vee wp(\alpha;\gamma, R) \equiv$

$S \rightarrow wp(\alpha;\beta, R) \wedge \neg S \rightarrow wp(\alpha;\gamma, R) \equiv$

$wp(S?;\alpha;\beta \cup \neg S?;\alpha;\gamma, R)$

Definition 5: $(S?;\alpha \cup (\neg S \wedge T)?;\gamma \cup (\neg S \wedge \neg T)?;\beta) \equiv S?;\alpha \cup \neg S?;(T?;\gamma \cup \neg T?;\beta)$

This rule may be used recursively to identify the meaning of a union of n programs $P_1?;\alpha_1 \cup P_2?;\alpha_2 \cup \ldots \cup P_n?;\alpha_n$. Also note that then restriction on such unions is exactly i) $P_1 \vee P_2 \vee \ldots \vee P_n$ is a tautology and ii) The P_i's are pairwise inconsistent. Any finite union satisfying i) and ii) is a PDDL-GC program.

Lemma 6:

i] $wp((P?;\alpha \cup \neg P?;\beta);(R?;\gamma \cup \neg R?;\delta), S) \equiv$

$wp((P \wedge wp(\alpha,R))?;\alpha;\gamma \cup (P \wedge wp(\alpha, \neg R))?;\alpha;\delta \cup$

$(\neg P \wedge wp(\alpha_1 R))?;\beta;\gamma \cup (\neg P \wedge wp(\beta, \neg R))?;\beta;\delta\ , S)$

ii] $wp(P?;(R?;\alpha \cup \neg R?;\beta) \cup \neg P?;(T?;\gamma \cup \neg T?;\delta\ , S) \equiv$

$wp(P \wedge R?;\alpha \cup P \wedge \neg R?;\beta \cup \neg P \wedge T?;\gamma \cup \neg P \wedge \neg T?;\delta\ , S)$.

The proof of lemma 6 is straightforward.

Lemma 7: Any PDDL-GC program formed by composition of base programs using $\cup$ and ; (without iteration) may be equivalently expressed in the disjunctive

[1] $wp(\alpha, \neg P) \equiv \neg wp(\alpha, P)$ if and only if α terminates. See Dijkstra [1].

normal form $P_1?;\alpha_1 \cup P_2?;\alpha_2 \cup \ldots \cup P_n?;\alpha_n$, where α_i is a concatenation $\beta^i_1; \beta^i_2; \ldots; \beta^i_{m_i}$ of base programs, and P_i is as in Definition 5.

Proof: Using lemma 4, we may drive base programs inside unions. Using 6 and 5, we may rewrite unions concatenated with unions and unions of unions into new unions. This is repeated until the disjunctive normal form is established. The new program is equivalent, at least in terms of input/-output behavior, because the wp's of each program are identical.

This result is obvious in the sense that, by expanding the computation tree of any program without iteration, there are only finitely many branches and thus there must a condition P_i that determines whether the computation proceeds down the i^{th} branch. However, in the process of proving lemma 7, we have provided an algorithm for calculating P_i. This is where the usefulness of the result lies.

Suppose and initial state, given by a set of predicate calculus axioms, is given and fixed. Then it is clear that, for any program α if $<\alpha> Q$ holds, then any iterations operate finitely many times and there will be a sequence of base programs $\beta_1, \ldots, \beta_k$ such that $<\beta_1; \ldots; \beta_k> Q$ is true. This yields:

Observation 8: In the STRIPS environment, where initial states are fixed, concatenation of base programs is sufficient to create a solution, if one exists.

Of course, iteration and union of base programs makes an interesting topic for A.I. research, but the STRIPS search method is sufficient. Because it is in general undecidable how many iterations are necessary to achieve a goal, this creates a difficult research issue, and is beyond the scope of this paper.

Theorems analogous to Lemmas 1 and 3 arise in the general union case.

Lemma 9: let $P_1 = wp\ (\alpha_1, R)$, $P_i = \neg P_1 \wedge \ldots \wedge \neg P_{i-1} \wedge wp\ (\alpha_i, R)$, $i=2, \ldots, n-1$ and $P_n = P_1 \wedge \ldots \wedge P_{n-1}$. Then if there are first order expressions $Q_1, \ldots, Q_n$ Such that $wp(Q_1?; \alpha_1 \cup Q_2?; \alpha_2 \cup \ldots \cup Q_n?; \alpha_n, R)$ is true, then $wp\ ((P_1?; \alpha_1 \cup \ldots \cup P_n?; \alpha_n), R)$ is true.

The proof is analogous to the proof of Lemma 1

Lemma 10: If $P_1, \ldots, P_n$ are chosen as in Lemma 9, then

$wp\ (P_1?; \alpha_1 \cup \ldots P_n?; \alpha_n, R) \equiv wp\ (\alpha_1, R) \vee wp\ (\alpha_2, R) \vee \ldots \vee wp\ (\alpha_n, R)$

Proof: It is easily demonstrated that

$wp\ (P_1?; \alpha_1 \cup \ldots \cup P_n?; \alpha_n, R) \equiv P_1 \rightarrow wp\ (\alpha_1, R) \wedge P_2 \rightarrow wp\ (\alpha_2, R) \wedge \ldots \wedge P_n \rightarrow wp\ (\alpha_n, R)$,

from definition 5. For all $i < n$, we have $P_i = \neg P_1 \wedge \neg P_2 \wedge \ldots \wedge \neg P_{i-1} \wedge wp\ (\alpha_i, R)$

Therefore for $i < n$, $P_i \rightarrow wp\ (\alpha_i, R)$ is clearly true, and thus

$wp\ (P_1?; \alpha_1 \cup \ldots \cup P_n?; \alpha_n, R) \equiv P_n \rightarrow wp\ (\alpha_n, R) \equiv \neg P_n \vee wp\ (\alpha_n, R)$

$\equiv \neg(\neg P_1 \wedge \neg P_2 \vee \ldots \wedge \neg P_{n-1}) \vee wp\ (\alpha_n, R) \equiv P_1 \vee P_2 \vee \ldots \vee P_{n-1} \vee wp\ (\alpha_n, R)$.

Observe, for sentences A and B, since $A \equiv (A \wedge B) \vee A$

$A \vee (\neg A \wedge B) \equiv A \vee (A \wedge B) \vee (\neg A \wedge B) \equiv A \vee ((A \vee \neg A) \wedge B) \equiv A \vee B$

Letting $A \equiv wp(\alpha, R)$ and $B = wp(\alpha_2, R)$, we have

$P_1 \vee P_2 \equiv wp\ (\alpha_1, R) \vee (\neg wp\ (\alpha_1, R) \wedge wp\ (\alpha_2, R)) \equiv wp\ (\alpha_1, R) \vee wp\ (\alpha_2, R)$.

Suppose for induction that $P_1 \vee P_2 \vee \ldots \vee P_k = wp\ (\alpha_1, R) \vee \ldots \vee wp\ (\alpha_k, R)$

This holds for $k = 2$. let $A = P_1 \vee \ldots \vee P_k$ and $B = wp\ (\alpha_{k+1}, R)$

Then $P_1 \vee \ldots \vee P_{k+1} \equiv (P_1 \vee \ldots \vee P_k) \vee P_{k+1} \equiv (P_1 \vee \ldots \vee P_k) \vee (\neg P_1 \wedge \ldots \wedge \neg P_k \wedge wp\ (\alpha_{k+1}, R))$

$\equiv (P_1 \vee \ldots \vee P_k) \vee ((\neg(P_1 \vee \ldots \vee P_k) \wedge wp\ (\alpha_{k+1}, R)) \equiv$

$(P_1 \vee \ldots \vee P_k) \vee wp\ (\alpha_{k+1}, R) \equiv wp\ (\alpha_1, R) \vee \ldots \vee wp\ (\alpha_k, R) \vee wp\ (\alpha_{k+1}, R)$

For any $k \leq n - 1$. Therefore letting $k = n - 1$, we have

$P_1 \vee \ldots \vee P_{n-1} = wp\ (\alpha_1, R) \vee \ldots \vee wp\ (\alpha_{n-1}, R)$.

Therefore $wp\ (P_1?; \alpha_1 \cup \ldots \cup P_n?; \alpha_n) \equiv P_1 \vee \ldots \vee P_{n-1} \vee wp\ (\alpha_n, R)$

$\equiv wp\ (\alpha_1, R) \vee \ldots \vee wp\ (\alpha_n, R)$ as desired.

Suppose we have an initial state I, a first order goal R, and some first order facts in our knowledge base which are true in all legal states. Denote the last set F. Facts about the state I are all facts concerning the assignment of terms to variables.

Now we might try to prove R from F, inserting knowledge of assignments of I. There is an algorithm, the Resolution Principle, [5], specifically designed for doing this. The Resolution Principle checks a first order expression R against a body of facts F by assuming $\neg R \wedge F$ and seeing if this is inconsistent. If $\neg R \wedge F$ is inconsistent, then F proves R, and R is true with respect to F. Resolution operates with two parts, one a theorem prover with respect to predicates, the other a unification principle that assigns terms to variables attempting to facilitate the theorem proving. In the DL context, unification changes states. Because we wish to control our states, and indeed unification might move us into an illegal state, we will operate resolution without the unification principle. If the unification principle is desired, it may be added as a base program, so there is no loss of generality from this alteration. In the process of resolution, we will, however, allow assignments of the state I to variables to be inserted for those variables. This limited unification, together with the theorem prover, might be termed resolution with respect to the state I. When we refer to resolution in a DL context, we will always mean resolution with I - unification.

Now in attempting to prove R from F inside I, we may operate resolution, attempting to prove a contradiction from $\neg R \wedge F \wedge I$, where I denotes both the state, and the first order characterization of that state (e.g. if $x \leftarrow e$ in I, then EQUAL (x,e) characterizes this assignment, where EQUAL (x,y) holds if and only if $x = y$).

If an inconsistency is found, the problem is solved and we need not apply programs, as R is already true. So suppose resolution operates for a long time and no contradiction is uncovered. Let A be an assertion proved by resolution. In the next section we shall give criteria for which expression A to choose. In the proof of A using resolution, a certain number of facts from I were used. Let this set be $B_1, \ldots, B_k$. These are called the state depenents facts in the proof of A. Finally, suppose R itself is a subgoal, $R \equiv wp(\alpha, Q)$ and α is a proposed program for making Q true (which may itself be a subgoal). So if R is true, then operation of α makes our desired result Q true.

Lemma 11: Suppose there is a program β such that $A \rightarrow wp(\beta, Q)$.
Then $wp((R?;\ \alpha \cup \neg R?;\ \beta), Q)$ is true.

Proof: $wp((R?;\ \alpha \cup \neg R?;\ \beta), Q) \overset{3}{\equiv} wp(\alpha, Q) \vee wp(\beta, Q) \equiv R \vee wp(\beta, Q)$
If R is true, then $R \vee wp(\beta, Q)$ is true. If $\neg R$ is true, then resolution proves A is true, which by hypothesis implies $wp(\beta, Q)$.

Lemma 12: Suppose there is a program β so that
$wp(\beta, R \vee (B_1 \wedge \ldots \wedge B_k \wedge \neg A))$ is true. Then
$wp(\beta;\ \alpha, Q)$ is true.

Proof: $wp(\beta, R \vee B_1 \wedge \ldots \wedge B_k \wedge \neg A) \equiv wp(\beta, R) \vee wp(\beta, B_1 \wedge \ldots \wedge B_k \wedge \neg A)$. is assumed true. If $wp(\beta, R)$ holds, then $wp(\beta;\ \alpha, Q) \equiv wp(\beta, wp(\alpha, Q))$ $\equiv wp(\beta, R)$ is true that is, operation of $\beta;\ \alpha$ proves Q.

On the other hand assume $wp(\beta, B_1 \wedge \ldots \wedge B_k \wedge \neg A)$ is true. We know that $\neg R \wedge F \wedge B_1 \wedge \ldots \wedge B_k \rightarrow A$, because $B_1, \ldots, B_k$ are precisely the state dependent facts used to prove A in the resolution operation. Since by hypothesis, after operation of β we have $B_1 \wedge \ldots \wedge B_k \wedge \neg A$, and because F is true in all states, we cannot have $\neg R$. Otherwise we would have $(\neg R \wedge F \wedge B_1 \wedge \ldots \wedge B_k) \wedge \neg A$ implying $A \wedge \neg A$. As a result, if $wp(\beta, B_1 \wedge \ldots \wedge B_k \wedge \neg A)$ is true, $wp(\beta, R)$ is true, which as we saw, implies $wp(\beta;\ \alpha, Q)$.

The interpretation of Lemma 11 is as follows: suppose we operate resolution trying to prove $wp(\alpha, Q)$ is true. We are unable to prove an inconsistency, but we do prove an assertion A. The assertion A is the difference between R and success. Consequently we attempt to handle this difference by adding a separate case. A sufficient condition, as we saw, for handling this difference is to find a program β so that $A \rightarrow wp(\beta, Q)$. The process is not contingent upon finding such a β, however. We might produce a candidate γ for β, and then extract the difference, handle this case, extract a new difference, etc.

It is in general undecidable if this method converges. Consequently, one can only note that, if there is a solution, and a backtracking method is implemented so that any possible concateration of base programs is eventually examined, then the procedure will terminate with a solution. If there is no solution, then in many instances this will be undecidable. Undecidability is a fact that we must live with, and do what we can. Of course, it is advisable to avoid trying to decide undecidable issues.

The reader should also note that there is no problem if resolution is stopped too early. For suppose $R \equiv wp(\alpha, Q)$ is true, and that resolution is stopped before an inconsistency arises. Then $wp((R?; \alpha \cup \neg R?; \beta), Q)$ will be true. This follows as R is true, and hence $R?; \alpha \cup \neg R?; \beta$ will execute α, and we assumed $R \equiv wp(\alpha, Q)$ is true. Thus there is no loss from terminating resolution too early.

The solution method inherent in Lemma 12 is perhaps more appropriate, as it relies on the concatenation and not disjunction of programs. As we observed in Observation 8, concatenation is a more promising approach, as we can always solve the problem via concatenation of base programs. In Lemma 12, A is again viewed as the difference between R and success. This time, instead of attempting to handle A as a separate case (as $A \rightarrow wp(\beta, Q)$, we have $\neg wp(\beta, Q) \rightarrow \neg A$, and $\neg A$ implies R true) instead, we attempt to find a program β to negate A. This would allow us to complete the proof of R, and thus have $wp(\beta; \alpha, Q)$ true.

Again we may note that even if resolution is terminated before it would have proved a contradiction, there is no loss of generality. The predicate $wp(\beta, R)$ is true so β cannot negate R. Consequently, $wp(\beta; \alpha, Q)$ remains true.

III. .2 A solution method

The basis of the solution method will be replications of Lemma 12. We initialize the problem by providing a goal R_o, an initial state I, a set of base programs, and a set of first order truths, F.

Begin the solution method by attempting to prove R_o, using resolution on F and state facts I. If R_o is proved, then it is currently true and there is no sense applying programs. If R_o is true, halt with this fact. Otherwise, extract the difference between R_o and success as in Lemma 12. The means of producing the difference A_o and the state dependent statements $B_1^o, \ldots, B_{k_o}^o$ will be discussed later. For now, assume we have them. Let $Q^o \equiv R_o \vee (\neg A_o \wedge B_1^o \wedge \ldots \wedge B_{k_o}^o)$. We select a program α_1, which is a candidate for making Q^o true. The means of producing this program, together with the mechanism for the choice of A, will be discussed later. For now, presume that we obtain it. Let $R^1 \equiv wp(\alpha_1, Q^o)$. Observe that lemma 12 implies that if R^1 is true, we are done.

The selection of R^1 will serve as initialization for a general induction to show that the solution method preserves the situation that R^i true means a solution has been found. Suppose we have found $R^n \equiv wp(\alpha_n, Q^{n-1})$, and if R^n is true, the problem is solved. Operate resolution on R^n to see if R^n is true. If we fail to prove R^n, extract a minimal difference A_n and required state dependent statements $B_1^n, \ldots, B_{k_n}^n$ in the proof of A_n. Let $Q^n \equiv R^n \vee (A_n \wedge B_1^n \wedge \ldots \wedge B_{k_n}^n)$, and choose a program $\alpha_n + 1$, to be a candidate for making Q^n true. Let $R^{n+1} \equiv wp(\alpha_{n+1}, Q^n)$. Observe that lemma 12 implies if R^{n+1} is true, then operation of α_{n+1} makes R^n true, and we assumed R^n true meant we had the problem solved. As a result, we have preserved the property that R^i true means the problem is solved. Thus, this is a valid recursion on the R^i's, moreover, if we reach a state where R^i is true, then operation of $\alpha_i; \alpha_{i-1}; \ldots; \alpha_2 ; \alpha_1$ makes R_o true, our goal. Hence, the desired program is $\alpha_i ; \alpha_{i-1}; \ldots ; \alpha_1$, by iteration of Lemma 12. This is the basis of the decision procedure. What is left is the formulation of the choices of A_n and α_{n+1}.

There is a natural procedure to guide the choice of α_n. Because a program only assigns terms to variables, although perhaps in quite complex ways, the choice of α_n is entirely one of finding a program that gives values to the variables which force $\neg R_n \wedge F$ to be inconsistent and hence make R_n true. Thus, we wish to find a program that unifies the literals in a set of expressions, that is, our program must simulate the most general unifier of the resolution principle. Because the most general unifier is readily calculable, it serves as a basis for the choice of α_n. Unification defines the variables which must be altered and even the manner in which they should be altered in order to prove $\neg R_n \wedge F$ inconsistent. Thus, a natural procedure arises for the choice of α_n.

This simulation of unification in the choice of programs is not arbitrary. Resolution is a special case of the problem solving environment described herein. In Resolution, there is a set of facts and a goal R, to be shown as a theorem of these facts. Resolution operates by literal elimination, or attempting to show R directly, and appropriate unification, or assignment of terms to variables. Resolution is the special case where all programs, or assignments of terms to variables, are permitted. In this Resolution environment, the most general unifier serves as a good method of proving R. When we restrict the set of valid programs and move to the problem solving environment of this paper, naturally the best choice of programs will emerge from simulating the unconstrained, resolution situation. Thus, the solution method of this paper yields a model of problem solving in a more general environment where Resolution is a special case.

The choice of A_n emerges from the Resolution model in a similar fashion. A_n is simply the expression which, when unified, proves a contradiction, or comes as close as can be to doing so. That is, A_n is the set of expressions the Resolution principle would choose to unify and solve the problem with.

The simulation of Resolution can be considered inefficient in the sense that it provides no information toward avoiding blind alleys. The wp predicates yield some information, but unless there is a means of evaluating where the weakest pre-condition, $wp(\alpha, R)$ is closer to the initial state than R itself, we will have no way of knowing if we are headed in the correct direction. As a result, we will define such a measure of "closeness to the goal" in the section on pseudometrics, which will complete the problem solving system of this paper.

As described, this system contains no backtracking. The skeleton of a backtracking system will be discussed, with the details left to the reader. Suppose a level n is reached so that no program eliminates a literal from $\bar{A}_n$. Then we should backtrack to level $n-1$ before α_{n-1} is applied and use the next best program α_{n-1}' instead of α_{n-1}, and then continue with the program synthesis. If at some point this precipitates backtracking to level $n=o$, we can either choose a new difference statement A_o', or we can begin looking at pairs (then triples, etc) of programs $\alpha;\beta$ to negate A_o. By observation 8, such a method, as it would eventually check all conjunctions of base programs, will be complete. This means if a program can be found to satisfy the goal, it will be found. As we mentioned, if there is no solution then this fact will often be undecidable.

It is probable that a combination of lemmas 11 and 12 would provide the optimal search plan. The general difficulty in finding a β so that $A \rightarrow wp(\beta, R)$, coupled with the question of which route (11 or 12) to take at any juncture prevents this from being developed here. It is a good topic for future investigation in this area.

III. 3 Comparison to The Resolution Principle

The final observation of this section is that in an environment with only the unification program, the system devolves to be a modification of resolution. First note that in the absence of other programs, we cannot change states. Therefore there are no state dependent facts. In order to prove an assertion R, the initial step was to operate resolution and then exit with either a contradiction and success, or a minimal assertion A_o and a set of state dependent facts. As noted, this set is empty, so we have only the assertion A_o.

Since there is only the unification program α_u, we set up a new goal wp $(\alpha_u, R \vee \neg A_o) \equiv R \vee \neg A_o$ and attempt to prove that. This is equivalent to choosing a conjunct of the set of theorems proved by resolution and specifically trying to negate it, as opposed to generally searching for a contradiction. If we fail to prove $\neg A_o$ (and thus R) with resolution, we take a second difference A_1 and try to prove $R \vee \neg A_o \vee \neg A_1$. This will continue until either we prove a contradiction or we get a new conjunct A_2. Because we will be taking progressively more minimal statements A_i, we are imitating resolution, only with a focus. We focus in on statements that seem the easiest to contradict, as opposed to generally resolving all the theorems of resolution. With backtracking, (the choosing of a different A_o), this method is approximately equivalent to the Resolution Principle.

IV. .1 Comparison to STRIPS

The GC problem solver described shares many features with STRIPS. As with the GC problem solver, STRIPS intially operates resolution to try to prove its goal. If this fails, it chooses a theorem T resulting from the resolution and tries to prove it. The theorem T corresponds to $A \wedge B_1 \wedge . . . B_k$ in our model. It is a result of the proof which, if made true, forces the goal to be true. STRIPS then searches for a satisfactory operator, as does the GC problem solver. Given this operator, it again tries to prove the new goal. If this fails, it extracts the difference of the new goal and the current state, and continues. This is precisely what the GC problem solver does, albeit in a more formal manner. Consequently, in a STRIPS problem environment, the GC solver simulates STRIPS.

However, there are several differences. The first major difference is that the GC problem solver has the ability to use information about the result of applying operators without actually applying them, in the form of the weakest precondition. This allows it some vision as to what branch is promising in the search tree, vision that is denied STRIPS. STRIPS with MACROPS attempts to cope with this lack of vision by remembering what was previously a successful path, however, there is no problem in doing the similar thing with the GC planner and STRIPS still lacks any vision concerning the application of concatenations of MACROPS. Thus, even in a STRIPS environment, the GC solver can check whether $wp(\alpha, R)$ is harder to satisfy than R, and hence whether to apply α.

A second difference regards unification. It may be that we declare some assignments of variables illegal and wish to prevent their occurance. STRIPS places no restrictions on unification in its operation, while such limitations are possible in DL. If we wish to rule out $x=0$ in our system, we may merely prevent any operator from ever assigning 0 to x.

This is one way in which GC has a wider scope than STRIPS.

Another important way is that in GC, we can call relevant data (a data program is simply $x \leftarrow e$, e being the data on variable x) with a simple program. Moreover we have programs available which are much more complicated than STRIPS operators.

Consequently, we see that the GC problem solver, while simulating STRIPS within a STRIPS environment, can calculate the results of its actions before it takes them, allowing it a modicum of "intuition" about which approach to take. Moreover, it is at home in more complicated environments where calculations and not just theorem proving must be done. Finally, by Lemma 2.12, we actually were able to prove that STRIPS did what it claimed, that is, if it found its subgoal true, then the sequence of operators did make the goal true.

IV 4.2 Pseudometrics

The search for an appropriate program to apply toward a goal may be directed using the method of pseudometrics [4].

A pseudometric is a distance measure over a space $\underline{\bar{X}}$. It is defined similarly to the metric of real analysis.

Definition: A pseudometric on a set $\underline{\bar{X}}$ is a function $p: \underline{\bar{X}} \times \underline{\bar{X}} \to R^+$ satisfying

i) $p(x,x) = 0$ for all $x \in \underline{\bar{X}}$

ii) $p(x,x') = p(x',x) \geq 0$ for all $x, x' \in \underline{\bar{X}}$

iii) $p(x, x') + p(x',x'') \geq p(x,x'')$ for all $x, x', x'' \in \underline{\bar{X}}$

The difference between a metric and a pseudometric is that we may have $p(x,x') = 0$ for $x \neq x'$ if p is a pseudometric but not if p is a metric. Because $\underline{\bar{X}}$ will be a space of models in our application and we do not wish to distinguish between every model with our measure, we will use pseudometrics.

We do not wish to distinguish between all models because often there will be characteristics which are irrelevant. If we do wish to distinguish all models, then metrics are relevant, and they are special cases of pseudometrics.

We will characterize the space of models by the first order sentences that are true in the model. Let $S_1, \ldots, S_n$ be a finite set of sentences, and $c_1, \ldots, c_n$ be nonnegative real numbers called criticality values. c_i is interpreted to be the "weight" or importance of the i^{th} sentence. Let M^j, M^k be two models. A clash set Γ_{jk} is defined by

$\Gamma_{jk} = \{i/S_i$ is consistent with one of the models and not the other$\}$.

A pseudometric p is then defined by

$$p(M^j, M^k) = \sum_{i \in \Gamma_{jk}} c_i$$

for any models M^j, M^k. To prove p is a pseudometric, we must establish that the rules i), ii) and iii) hold.

i) if $M^j = M^k$, then S_i is either consistent with both or neither. Therefore $\Gamma_{jk} = \emptyset$ and $p(M^j, M^k) = \sum_{i \in \Gamma_{jk}} c_i = \sum_{i \in \emptyset} c_i = 0.$

ii) Observe that $\Gamma_{jk} = \Gamma_{kj}$ as the order of the models is irrelevant.

Therefore

$$p(M^j, M^k) = \sum_{i\in\Gamma_{jk}} c_i = \sum_{i\in\Gamma_{kj}} c_i = p(M^k, M^j)$$

moreover, since $c_i \geq 0$, $\sum c_i \geq 0$.

iii) Finally given M^o, M and M^2 claim $\Gamma_{02} \subseteq \Gamma_{01} \cup \Gamma_{12}$

Proof of claim: Suppose $i\in \Gamma_{02}$. S_i is either consistent with M^o and not with M^2, or with M^2 and not M^o. Suppose the former holds. S_i is either consistent with M^1 or not. If S_i is consistent with M^1, we have that it is inconsistent with M^2 and thus $i\in\Gamma_{12}$. Otherwise its inconsistent with M^1 and not with M^o, and consequently $i\in \Gamma_{01}$. The argument where S_i is inconsistent with M^o and consistent with M^2 is symmetric. Therefore $i\in \Gamma_{02}$ implies $i \in \Gamma_{01} \cup \Gamma_{12}$, and $\Gamma_{02} \subseteq \Gamma_{01} \cup \Gamma_{12}$, as desired.

Therefore $p\,(M^o, M^1) + p\,(M^1, M^2) = \sum_{i\in\Gamma_{01}} c_i + \sum_{i\in\Gamma_{12}} c_i \geq \sum_{i\in\Gamma_{01}\cup\Gamma_{12}} c_i$

$$= \sum_{i\in\Gamma_{02}} c_i + \sum_{i\in(\Gamma_{o1}\cup\Gamma_{12})/\Gamma_{02}} c_i \quad \geq \quad \sum_{i\in\Gamma_{02}} c_i = p(M^o, M^2),$$ and

consequently the triangle inequality holds.

Thus we have shown that the construction of p satisfies the properties of a pseudometric. Thus p is a distance measure over the space of models. The distance of M^o and M^1 as measured by p represents the clash of those models with respect to the sentences $S_1, \ldots, S_n$ and the weights c_i associated with them. The optimal use of p requires the appropriate choice of S_i and c_i. We will discuss this in the next section.

IV. .3 Directing a problem solver

Suppose we have a goal model in mind M_G and a set of sentences $S_1, \ldots S_n$ which are the criteria by which we judge a model. Further suppose the weights $c_1, \ldots, c_n$ are given. If M^0, M^1 are two achievable models, and if

$p(M^o, M_G) > p(M^1, M_G)$, then by the criteria of the pseudometric, M^1 is closer to M_G than M^o is. This is the sense in which we mean a pseudometric can guide a problem solver. STRIPS implicitly assumes a pseudometric where $c_i = 1$ for all $i \leq n$. The sentences S_i are the negations of the assertions in the goal model. This is easily seen. When STRIPS checks if a model M is close to a goal model M_G, it checks whether the assertions of the goal model are proved in M. This is equivalent to the negations of the assertions of the goal model being inconsistent with M, since the negation of an assertion of a goal model is inconsistent with the goal model. This is equivalent to checking if $I \notin \Gamma$, as $i \notin \Gamma$ if S_i is inconsistent with M (and S_i is inconsistent with M_G, S_i being the negation of an assertion of M_G). Since STRIPS does not differentiate the weight of the different sentences, we may assume $c_i = 1$. In passing, we note that ABSTRIPS [6] assigns different weights to the sentences S_i and gains some direction as a result [4].

The notion of inconsistency applied is derived from standard proof theory, using modus ponens and universal generalization. In particular, the Resolution Principle would yield the same notion.

It is the choice of S_i and c_i that concerns us here. First we identify the S_i. Recall that we have a subgoal (goal) R which we would like to use programs to make true in the initial state. That is, we would like R to be true, or to find a program β so that wp (β, R) is true, in the goal model. If the model M is a true representation of the state of the world, then we would like $p(wp(\beta,R), M) = 0$ to mean wp (β,R) is true in the initial state M. Thus we want no assertion of wp (α,R) to be inconsistent with M. Consequently let $S_1, \ldots, S_n$ be the assertions of M. Then $p(wp(\beta,R),M) = 0$ if and when wp (α,R) is not inconsistent with M. Therefore, we wish to let $S_1, \ldots, S_n$ be the assertions of the initial state, the facts that are known to be true.

The choice of c_i's are a more difficult problem. However, there are some heuristics to consider in their choice.

First, if there is a program β_ℓ so that for almost any R, S_i is consistent with wp(β_ℓ, R), then c_i should be low. This is because we can apply β_ℓ with low expected complications so that S_i becomes only a minor problem. On the other side, if most of the programs β are such that S_j is inconsistent with wp(β,R) for most R, then C_j should be large. This follows because C_j is determined by what complications it presents in the truth of wp(β, R)

The best choice of the criticality values depends on the problem environment. Obviously, the relative importance of aspects of a problem is a subjective issue, and developing a more formal theory is a topic for future research.

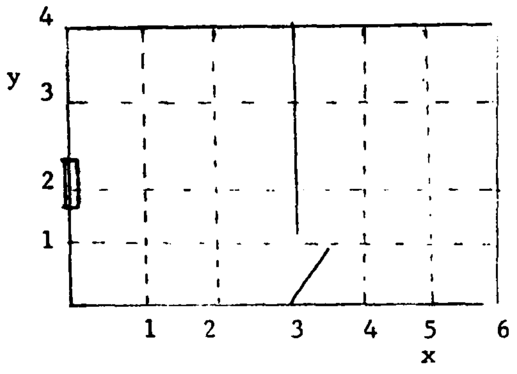
IV. .4 Application of a pseudometric

Once a pseudometric has been constructed, we may use it to direct the choice of which program to apply next. In particular, we may define a greedy strategy, one that chooses the maximum decrease in the pseudometric value with each program application. If backtracking is available, this is perhaps as good a method as any. To use the pseudometric to guide the program search, we must use it as the criteria to judge which program to choose. This requires evaluating the metric at each time a candidate program is proposed in the manner described. Once such a metric is set up, it is a tremendous asset, as it defines the notion of distance to a solution. It should be emphasized that much more general pseudometrics may be instituted [see example in McAfee & Whinston 1980] than the one described in this paper. However, a formal notion of distance to a goal is a pseudometric, and if a machine is ever to act as if it knew what 'close' means, it must have a measure of distance.

For computers to solve general problems, it is likely that they will have to simulate human decision making. Despite vast computational power and extreme speed per computation, computer solutions to general problems are still unacceptably slow. It is thus of interest that the procedure described in this paper is analogous to a human method of problem solution, at least in the abstract. The procedures first checks if there is anything to be done, that is, if the goal is true currently. Its next step is to consider the possible alternatives. Analogous to a person, it checks if the alternative (program) does something potentially useful, by means of its pseudometric intuition, and whether the preconditions of the alternative are reasonable, using the weakest preconditions of Dijkstra. Backtracking gives us the generality of these methods, and the pseudometric gives us the means of telling a computer what is reasonable and what is not. As a result, the methods of this paper provide a new approach to automated problem solving.

V. Example

The example we provide is similar to that of artificial intelligence literature. We have a robot in a 2 room suite. The robot's position POSITION (x,y) is described by the variables (x,y). There is a window which is open when V = 1 and shut if V = -1. The variable V is associated with the predicate "WINDOW (V)." There is a door between the rooms, which can be open $\ell = 1$ or shut $\ell = -1$, and ℓ is associated with the predicate OPEN (ℓ). There are 3 operators given below.



β_1:	OPENWINDOW	Precondition:	POSITION (0,2)
		Addition:	WINDOW (1)
		Deletion:	WINDOW (-1)
β_2:	THRUDOOR	Preconditions:	OPEN (1) $\wedge$ POSITION (2,1)
		Addition:	POSITION (4,1)
		Deletion:	POSITION (2,1)
β_3:	INROOMMOVETO (x_2, y_2)	Preconditions:	POSITION $(x_1,y_1) \wedge [(x_1 \le 3 \wedge x_2 \le 3) \vee (x_1 > 3 \wedge x_2 > 3)]$
		Addition:	POSITION (x_2,y_2)
		Deletion	POSITION (x_1,y_1)

The corresponding GC programs are

β_1: $(x=0 \wedge y=2)?; V \leftarrow 1 \cup \neg(x=0 \wedge y = 2)?;$ ABORT

β_2: $(\ell=1 \wedge x=2 \wedge y=1)?; x \leftarrow 4; y \leftarrow 1 \cup \neg(\ell=1 \wedge x=2 \wedge y=1)?;$ ABORT

β_3: $[(x\leq 3 \wedge x_2\leq 3) \vee (x>3 \wedge x_2>3)]? \; x \leftarrow x_2; \; y \leftarrow y_2 \cup \neg[(x\leq 3 \wedge x_2\leq 3) \vee (x>3 \wedge x_2>3)]?;$ ABORT

We denote the state changing parts of β_i by $\hat{\beta}_i$, i.e.,

$\hat{\beta}_1 \equiv V \leftarrow 1$, $\hat{\beta}_2 \equiv x \leftarrow 4; y \leftarrow 1$, $\hat{\beta}_3 \equiv x \leftarrow w; y \leftarrow z$.

Our goal for this example is the following model R_o:

POSITION (6,0) $\wedge$ WINDOW (1)

or, in DL notation, $x = 6$, $y = 0$, $V = 1$.

Our initial state, M_I, in this example is:

POSITION (0,4) $\wedge$ WINDOW (-1) $\wedge$ OPEN (1)

That is, $x = 0$, $y = 4$, $V = -1$ and $\ell = 1$.

According to the method of this paper, to calculate the pseudometric we take the sentences of the initial model, negate them to form the critical sentences $S_1, \ldots, S_n$, and assign them criticality values $c_1, \ldots, c_n$. We shall do this with the sentences $V = -1$ and $\ell = 1$, however, we shall introduce an interesting twist on the sentence $x = 0 \wedge y = 4$. Essentially this will involve using the broken euclidean distance metric as the measure of distance.

Let S_1 be $\neg(V = -1)$ and S_2 be $\neg(\ell = 1)$. Let the criticality values associated with S_1 and S_2 both be unity. In addition, define $c_3\ (x_1, y_1, x_2, y_2)$

$$c_3\ (x_1, y_1, x_2, y_2) = \begin{cases} \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} & \text{If } (x_1\leq 3 \wedge x_2\leq 3) \vee (x_1>3 \wedge x_2>3) \\ \sqrt{(x_1 - 4)^2 + (y_1 - 1)^2} & \text{If } (x_1>3 \wedge x_2\leq 3) \\ +2 + \sqrt{(x_2 - 2)^2 + (y_2 - 1)^2} & \end{cases}$$

It is readily verifies the c_3 gives the broken Euclidean distance metric for the distance between (x_1, y_1) and (x_2, y_2). (see diagram 2)

Now we define, for two models M^1 and M^2

$$p(M^1, M^2) = \sum_{i\in\Gamma_{12}} c_i + c_3(x_1, y_1, x_2, y_2)$$

where POSITION (x_1, y_1) is true in M^1 and POSITION (x_2, y_2) is true in M^2. e_3 measures the distance, which is a metric and hence a pseudometric, to one position from another.

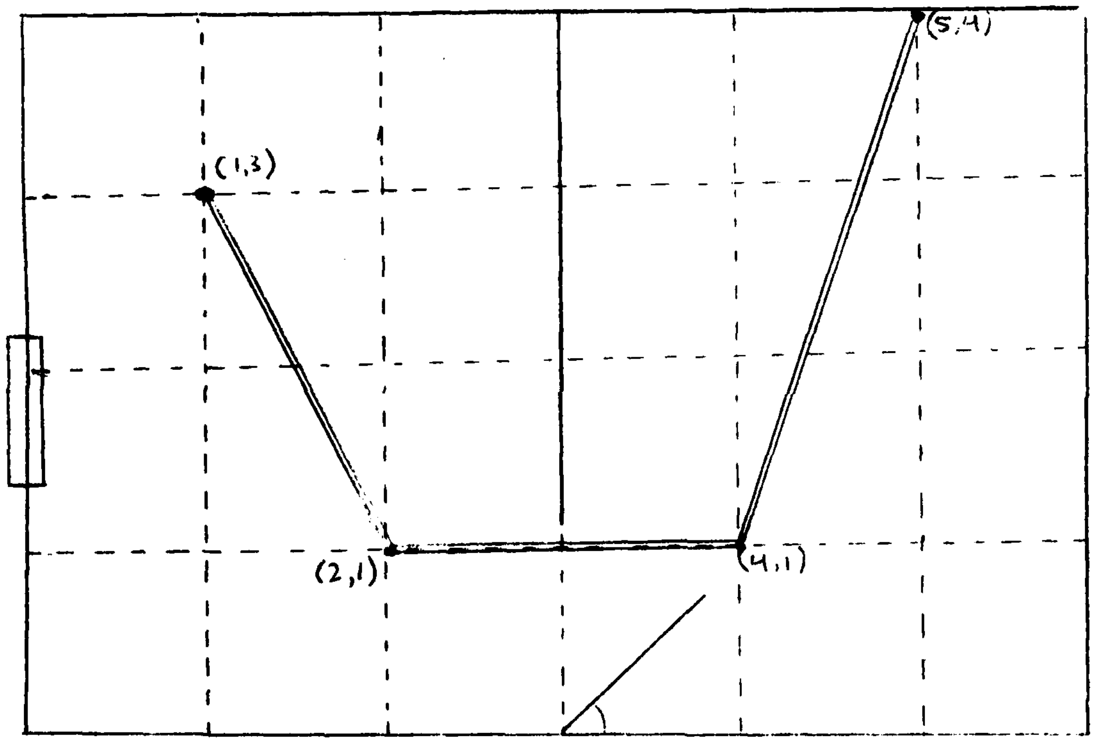


Diagram .2 Distance through the door.

distance from (1,3) to (5,4) is the distance from (1,3) to (2,1) $(\sqrt{(1-2)^2 + (3-1)^2} = \sqrt{5})$ plus the distance from (4,1) to (5,4) $(\sqrt{(4-5)^2 + (1-4)^2} = \sqrt{10})$ plus the distance from (2,1) to (4,1), or 2.

In this example, the candidate programs are chosen fortuitously. The actual execution would require several backtrackings. The point of this example is to illustrate the use of weakest preconditions and pseudometrics.

Observe, as V is disagreed on

$$p(M_I, R_0) \equiv \sum_{i \in T} c_i + c_3 = c_1 + c_3(0, 4, 6, 0)$$

$$= 1 + \sqrt{(6-4)^2 + (0-1)^2} + 2 + \sqrt{(0-2)^2 + (4-1)^2}$$

$$= 3 + \sqrt{5} + \sqrt{13}.$$

Any STRIPS type program β has a special form, which is given by $P?;\hat{\beta} \cup \neg P?;ABORT$.

For any sentence S, $wp(\beta,S)$ also has a special equivalent form, as we now show.

$wp(\beta,S) \equiv wp(P?;\hat{\beta} \cup \neg P?ABORT, S) \equiv$

$(P \rightarrow wp(\hat{\beta}, S)) \wedge (\neg P \rightarrow wp(ABORT, S)) \equiv$

$(\neg P \vee wp(\hat{\beta},S)) \wedge (P \vee false) \equiv (\neg P \vee wp(\hat{\beta},S)) \wedge P \equiv$

$P \wedge wp(\hat{\beta},S)$.

We shall use this simple reduction throughout the example.

Let β_3 be our first candidate program. It is a natural choice as the preconditions of β_1 and β_2 are not satisfied. The algorithm yields:

$r_1 \equiv wp(\beta_3, R_0) \equiv [(x \leq 3 \wedge x_2 \leq 3) \vee (x > 3 \wedge x_2 > 3)] \wedge wp(\hat{\beta}_3, R_0)$

$\equiv [(6 \leq 3 \wedge x_2 \leq 3) \vee (6 > 3 \wedge x_2 > 3)] \wedge wp(x \leftarrow x_2; y \leftarrow y_2, R_0)$

$\equiv [x_2 > 3] \wedge POSITION\ (x_2, y_2) \wedge WINDOW\ (1)$, as $6 \not\leq 3$.

$p(M_I, R_1) = \sum_{i \in T} c_i + \sqrt{(0-2)^2 + (4-1)^2} + 2 + \sqrt{(4-x_2)^2 + (1-y_2)^2}.$

minimizing $p(M_I, R_1)$ with respect to x_2, y_2 yields $x_2 = 4$ and $y_2 = 1$

and R_1 = Position (4, 1) $\wedge$ WINDOW (1)

and hence $p(M_I, R_1) = 3 + \sqrt{13}$, a distinct gain.

The preconditions of β_1 are not satisfied, and the use of β_3 will not provide any reduction of p, so we let β_2 be our next candidate program. Then we have:

$R_2 \equiv wp(\beta_2, R_1) \equiv (\ell = 1 \wedge x = 4 \wedge y = 1) \wedge wp(\hat{\beta}_2, R_1)$

$(x = 2 \wedge y = 1) \wedge wp\ (x \leftarrow 4; y \leftarrow 1, POSITION\ (4, 1) \wedge WINDOW\ (1))$

$\equiv$ POSITION (2, 1) $\wedge$ WINDOW (1). We have

$p(M_I, R_2) = c_1 + c_3 = 1 + \sqrt{(0-2)^2 + (4-1)^2} = 1 + \sqrt{13},$

again an improvement.

The next program we check is β_3.

$R_3 = wp\ (\beta_3, R_2) \equiv x = x_2 \wedge y = y_2 \wedge x < 3 \wedge V = 1$

$p(R_3, M_I) = 1 + \sqrt{(0-x_2)^2 + (4-y_2)^2} = 1 + \sqrt{x_2^2 + (4-y_2)^2}$

we could minimize this, but it would not deal with opening the window. Consequently the machine would make an error here (going to (0,4)). At this point p = 1 and the machine would attempt to open the window by applying the operator β_1. This reveals the position problem as we see

$R_4 \equiv wp(\beta_1, R_3) \equiv (x = 0 \wedge y = 2) \wedge wp(\hat{\beta}_1, R_3)$

$\equiv (x = 0 \wedge y = 2) \wedge (x = x_2 \wedge y = y_2) \wedge V = -1$

$\equiv x = 0 \wedge y = 2 \wedge V = -1$ Thus the wp's direct $x_2 = 0 \wedge y_2 = 2$

$p(R_4, M_I) = \sqrt{(0 - 0)^2 + (2 - 4)^2} = 2$, an improvement.

Finally, we operate β_3 once more

$R_5 \equiv (\beta_3, R_4) \equiv x = x_3 \wedge y = y_3 \wedge wp(\hat{\beta}_3, R_4)$

$\equiv x = x_3 \wedge y = y_3 \wedge [x = 0 \wedge y = 2 \wedge x < 3] \wedge (V = -1)$

$\equiv x = x_3 \wedge y = y_3 \wedge x \le 3 \wedge V = -1$

Minimizing $p(R_5, M_I) = \sqrt{(0-x_3)^2 + (4 - y_3)^2}$

subject to $x_3 \le 3$

yields $x_3 = 0, y_3 = 4$ and $p(R_5, M_I) = 0$.

$p(R_5, M_I) = 0$ means the problem is solved by the program sequence $(\beta_3; \beta_1; \beta_3; \beta_2; \beta_3)$.

There are two significant characteristics displayed by this example. First, the notion of a pseudometric admits such measures as Euclidean distance. One is not restricted to such pseudometrics as discussed in section 4, and we may use ordinary measures to measure the difference of models.

Perhaps more important, the pseudometric may be used to guide the choice of operators where an operator is a schema. In both the first and third application of β_3, we used minimization of the pseudometric as the choice function for the values (x_i, y_i), the point from which we were moving. As we saw in the second application of β_3, this is not an ideal guide (because it tells the machine to make easy gains) but it is a guide to the choice of values for a schema. Used in conjunction with the preconditions of the next program, it becomes a very useful guide. Backtracking is employed to handle cases where it fails to guide impeccably.

One final point, some pseudometrics arise naturally on a space of models. In our example, the distance measure c_3 is an appropriate measure due to considerations resulting from Euclidean space, and the existance of a barrier. Others, such as the measure of the value of a window discrepancy, are arbitrary. Their value may rest on the judgement of the individuals who initially describe the program.

In this example we chose the programs so that they worked. However, if we are faced with a problem not contrived as an example, we could use the pseudometric as a guide to problem solution. If the other programs were tried and then checked via the pseudometric for improvement, we would find that generally the best program would also minimize the pseudometric. This occurs because the pseudometric values were improvements and the best program generally results in improvement.

VI. Conclusion and research topics

In this paper, several extensions of problem solvers are examined. Specifically, we looked at problem solving in a larger language than first order calculus, one that incorporates the programs which form the nucleus of problem solving systems. Next, a method of enbedding a portion of this language in the first order calculus was examined. We proposed a problem solving system that extends STRIPS and the Resolution Principle. We then formalized the notion of distance in models and showed how this may be used to direct the search of the machine for a solution.

Perhaps the most important research topic is the inclusion of program union and iteration in the program synthesis algorithm. These are important methods of combining programs, and present some complications in their inclusion. Lemma 2.11 shows how union may be profitably included. However, the correct use of this Lemma is difficult to incorporate into the algorithm. Iteration presents a tricker problem, as the weakest pre-condition is an infinite disjunction. Some promising research suggests that iterations could be called for when induction is called for. However, the problem of when to use it still remains.

With regard to pseudometrics, an important line of inquiry is the optimal utilization of the measure and minimizing the cost of evaluation of the pseudometric. Moreover the choice of the criticality values still remains open. Effective use of pseudometrics, and indeed, efficient decision procedures, requires additional research on this topic.

[1] Dijkstra, Edsger W. "Guarded Commands, Nondeterminacy and Formal Derivation of Programs, "Communications of the ACM, Vol. 18 No. 8, (1975) pp. 453-457.

[2] Fikes, Richard E. and Nils J. Nilsson. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence 2, 1971 pp. 189-208.

[3] Harel, D., First-Order Dynamic Logic, Lecture Notes in Computer Science, Vol. 68, Springer-Verlag, 1979.

[4] McAfee, R.P. and A.B. Whinston. "A Formal Model of Problem Solving," Journal of Policy Analysis and Information Sciences, forthcoming April 1980, Vol. 4, No. 2

[5] Robinson, J.A. "A Machine Oriented Logic based on the Resolution Principle," Journal of the ACM, Vol. 12, No. 1 (1965) pp. 23-41.

[6] Sacerdoti, E.D. "Planning in a Hierarchy of Abstraction Spaces," Artificial Intelligence 5, 1974 pp. 115-135.